CIGI QUALITA MOSIM 2023
Génération automatique des applicatifs de contrôle-commande pour des logiciels chiffrés

SORAYA MESLI-KESRAOUI, DJAMAL KESRAOUI

SEGULA Engineering France
Parc Technellys 165 Rue de la Montagne du Salut, 56602 Lanester Cedex, France
soraya.kesraoui@segula.fr, djamal.kesraoui@segula.f

***Résumé* – La génération automatique des applicatifs de contrôle-commande a été largement étudiée dans la littérature et plusieurs approches ont été proposées. Toutefois, ces approches manipulent directement des fichiers ouverts (en XML ou en texte) qui ne sont pas chiffrés. Aucune solution n'a été rapportée dans la littérature portant sur la génération des applicatifs de contrôle-commande pour des logiciels chiffrés.**

Dans cet article, une nouvelle solution permettant la génération automatique des applicatifs de contrôle-commande sur des logiciels fermés et chiffrés est proposée. Cette solution propose un flot de conception permettant l'utilisation des interfaces graphiques de ces logiciels pour la génération des applicatifs de contrôle-commande. En effet, avec le langage AutoHotKey, cette solution reproduit automatiquement toutes les actions de développement, réalisées par les développeurs industriels sur les interfaces graphiques des logiciels chiffrés pendant le développement d'un applicatif de contrôle-commande. Les expérimentations réalisées sur un système de traitement d'eau montrent que cette solution s'applique sur tous les logiciels qu'ils soient chiffrés ou non, et que ses performances de temps de calcul sont meilleures qu'une approche manuelle.

***Mots clés* – Contrôle-commande, génération automatique, Autohotkey.**

1 INTRODUCTION

La conception des systèmes de contrôle-commande est une activité de plus en plus complexe car elle fait intervenir plusieurs concepteurs, issus d'horizons techniques très variés. Cette variété de profils entraîne des difficultés de compréhension des spécifications, ce qui se traduit par des erreurs de conception détectées lors de la phase de test (Bignon, 2012).

Partant de ce constat, plusieurs solutions de génération automatique ont été proposées (Bato et al., 2017; Bignon, 2012; Bignon et al., 2010; Drath et al., 2006; Goubali et al., 2014; Gruner et al., 2014; Jbair et al., 2019; Qasim et al., 2020; Steinegger & Zoitl, 2012). La génération automatique réduit les temps de conception et garantit une cohérence entre les modèles d'entrée et de sortie. Ces solutions permettent la génération des applicatifs de contrôle-commande à partir des modèles métier. Concrètement, elles consistent à générer un fichier XML (Extensible Markup Language) ou un fichier texte à partir d'un autre fichier en appliquant un ensemble de règles de transformation. La génération automatique des applicatifs de contrôle-commande nécessite l'accès aux fichiers (XML ou texte) ouverts et donc non chiffrés.

Toutefois, certains constructeurs industriels adoptent des stratégies de chiffrement des différents logiciels industriels. Ce chiffrement est un frein à la génération automatique des applicatifs de contrôle-commande.

1.1 Motivation

Le but de ces travaux est de proposer une nouvelle solution pour la génération des applicatifs de contrôle-commande, comprenant une interface de supervision et un programme de commande, sur des logiciels chiffrés (les fichiers en XML ou en texte ne seront plus utilisés). Dans ce cadre, trois verrous sont à lever.

1. Sur des logiciels chiffrés, seules leurs données chiffrées ou leurs interfaces graphiques sont accessibles. Une première piste consiste à déchiffrer les données pour ensuite les exploiter. Cependant, les entreprises qui produisent ces logiciels ne diffusent pas leurs méthodes de chiffrement. L'opération de déchiffrement peut s'avérer impossible. D'autre part, certains algorithmes proposent de manipuler directement les données chiffrées. Toutefois, les opérations réalisées par ces algorithmes sont très réduites et ne peuvent pas être utilisées dans la génération des applicatifs. La seule piste possible est alors d'exploiter l'interface graphique du logiciel afin de générer des applicatifs.
2. Lors de la manipulation des interfaces graphiques, l'intégrité des données produites est un enjeu majeur. En effet, le but ici est de générer des applicatifs de contrôle-commande pour des systèmes complexes où la notion de sûreté de fonctionnement est très importante. Le défi ici est de générer des applicatifs fonctionnels, fiables et sûrs de fonctionnement (sans erreurs).
3. D'autre part, l'utilisation des interfaces graphiques pour la production de données engendre un problème de généricité de la solution proposée et des difficultés techniques. En effet, la génération automatique pour un logiciel cible dépend de la structure de son interface graphique. Comme les interfaces graphiques des logiciels sont différentes, le défi est de généraliser la solution pour étendre son applicabilité sur des interfaces graphiques différentes.

1.2 Contribution

La littérature sur la génération des applicatifs de contrôle-commande pour des logiciels chiffrés est inexistante. Les solutions existantes (Steinegger et Zoitl 2012; Drath et al. 2006; Gruner et al. 2014; Jbair et al. 2019; Qasim et al. 2020; Bato et al. 2017) produisent des données non-chiffrées (XML ou texte) et ne peuvent pas être appliquées sur des logiciels chiffrés. Dans cet article, nous présentons une nouvelle solution pour la génération des applicatifs de contrôle-commande sur des logiciels chiffrés. Nos contributions sont comme suit :

1. Cette solution repose sur l'utilisation des interfaces graphiques des logiciels pour reproduire d'une manière automatisée toutes les actions (clics de souris, frappe de clavier) nécessaires à la création d'un programme de commande et d'une interface de supervision.
2. Pour garantir la généricité de notre solution, un flot de conception a été proposé. Ce dernier permet d'organiser la génération en deux étapes : étapes de construction d'une bibliothèque et une étape d'instanciation des composants sur les logiciels chiffrés.
3. Pour la construction de la bibliothèque, les techniques d'enregistrement et de rejeu, utilisées efficacement dans les approches de programmation par démonstration (Goubali, 2017), ont été utilisées. En effet, l'enregistrement permet de créer les composants de la bibliothèque et le rejeu permet son importation dans le logiciel cible.
4. Pour l'étape d'instanciation, des modules ont été développés permettant d'instancier tous les composants sur des programmes de commande et également sur l'interface de supervision.

1.3 Organisation de l'article

La section 2 présente un état de l'art lié à notre étude. Nous présentons dans la section 3, l'approche générale proposée pour la génération des applicatifs de contrôle-commande chiffrés ainsi que ses différentes étapes. En section 4, l'évaluation de notre approche sur un système de traitement d'eau est présentée et les résultats sont discutés. Nous terminons par une conclusion et les perspectives de ce travail dans la section 5.

2 ETAT DE L'ART

2.1 Génération automatique des programmes de contrôle-commande

La génération automatique des programmes de commande et des interfaces de supervision a été largement traitée dans la littérature. Les approches de génération utilisées sont de deux types (Koziolek et al., 2020) : la génération basée sur les règles et la génération basée sur les modèles en utilisant l'IDM.

2.1.1 Génération basée sur les règles

La génération basée sur les règles consiste à générer des programmes de commande à partir de règles, définies dans une syntaxe précise. Ces règles sont spécifiques à un domaine et permettent d'enregistrer les connaissances d'un domaine sous la forme d'un ensemble de règles réutilisables (Gruner et al., 2014). Dans (Drath et al., 2006; Steinegger & Zoitl, 2012) par exemple, les auteurs génèrent des codes de commande à partir des règles de contrôle de verrouillage (interlock control en anglais).

2.1.2 Génération basée sur les modèles

La génération basée sur les modèles consiste à générer des programmes de commande à partir de modèles plus abstraits ou des modèles métiers en utilisant les concepts de l'ingénierie dirigée par les modèles (IDM) (Combemale, 2008). Au niveau des programmes de commande, les travaux de (De Lamotte, 2006) proposent une solution pour la modélisation, l'analyse et la génération automatique des programmes de commande pour des systèmes reconfigurables. Pour ce faire, l'auteur propose un langage de haut niveau (abstrait) permettant la modélisation de l'architecture physique et logique du système. À partir de ce modèle, les programmes de commande (SFC) sont alors générés automatiquement par transformation de modèles. Sacha (2010) propose une approche pour la génération automatique des programmes de commande à partir d'une spécification abstraite en UML. Les auteurs de (Wang et al., 2009) utilisent le langage formel des automates temporisés pour modéliser le comportement d'un programme de commande. À partir des automates temporisés, les codes de programmes de commande sont générés en langage Ladder. Les auteurs de (Jbair et al., 2019) proposent de générer automatiquement des codes de commande à partir des modèles virtuels.
Au niveau des IHMs, dans (Qasim et al., 2020) une approche basée sur les modèles a été proposée pour la génération des IHM de supervision à partir d'une description abstraite de l'IHM en UML. Dans (Bato et al., 2017), une approche de génération automatique des interface de supervision a été proposée en utilisant les concepts de l'IDM. L'interface est initialement décrite à travers un ensemble de Template. Ces derniers sont ensuite utilisés pour la génération de l'interface exécutable.

2.1.3 Le projet Anaxagore

En se basant sur l'ingénierie dirigée par les modèles (IDM), le projet Anaxagore (Bignon 2012; Bignon et al. 2013; Bignon et al. 2010) offre une solution pour la génération conjointe d'un programme de commande et d'une interface de supervision à partir d'un P&ID et d'une bibliothèque de composants standards (Figure 1).
Le P&ID est un schéma structuro-fonctionnel constitué d'un ensemble de symboles et de connexions relatifs à l'architecture du procédé et au fonctionnement d'un système. Un composant de la bibliothèque est l'unité constitutive du procédé du système. Le composant peut être relatif à du matériel (une vanne, une pompe...) ou à des fonctionnalités du système.

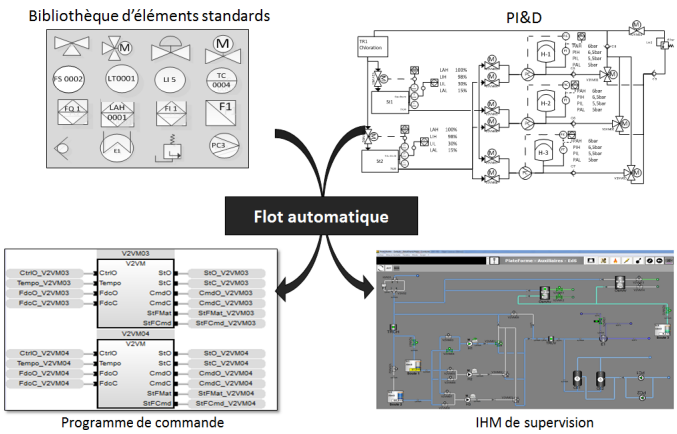

Figure 1. Génération automatique dans (Bignon, 2012)

La génération automatique commence par des transformations de modèles pour produire, à partir d'un P&ID (fichier XML créé sur Microsoft Visio), une nomenclature. La nomenclature est un ensemble d'items représentant chacun une instance d'un élément du système.
La génération des programmes de commande utilise les données de la nomenclature et de la bibliothèque des composants standards pour instancier les composants selon les données de la nomenclature.
La génération des interfaces de supervision nécessite en plus de la nomenclature, une liste d'équipotentiels (Bignon, 2012). Le potentiel d'un point du système, dans notre précédent cas d'usage, correspond à la pression du fluide en ce point en régime permanent (Figure 2). Un équipotentiel désigne une zone du système dont tous les points sont au même potentiel. Dans l'exemple d'un système d'eau douce sanitaire, l'eau douce envoyée par la soute est reçue avec la même pression par les vannes V2M04, V2M06 et V2VM08. La liste des équipotentiels du système est utilisée pour générer les différentes configurations du système.

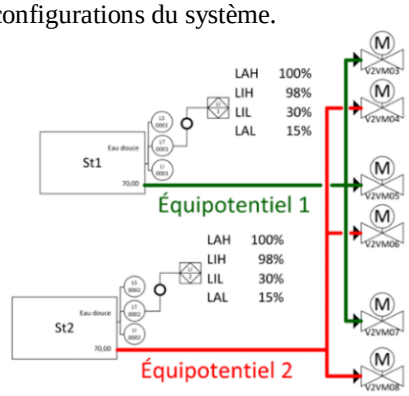

Figure 2. Exemple d'un équipotentiel

D'autre part, le concept d'IDM a permis la génération des applicatifs de contrôle-commande de haut niveaux ou globaux (Goubali 2017; Goubali et al. 2014; Goubali et al. 2016; Goubali, Girard, Guittet, Bignon, Kesraoui, Boulhic, et al. 2016).

2.1.4 Bilan

Toutes les approches citées précédemment, manipulent directement les données des applicatifs générés. Ces approches ne peuvent pas être appliquées pour la génération des applicatifs chiffrés dont les données sont chiffrées. Sur des logiciels chiffrés, seules leurs données chiffrées ou leurs interfaces graphiques sont accessibles. Ainsi, pour produire des données chiffrées pour ces logiciels, deux solutions existent : la manipulation directe de leurs données chiffrées ou de leur interface graphique.

2.2 Manipulation des données chiffrées

La manipulation des données chiffrées consiste à produire des données chiffrées sans les déchiffrer car les algorithmes de chiffrement utilisés par les logiciels ne sont pas accessibles. Certains algorithmes réalisent des calculs sur des données chiffrées sans les déchiffrer.
Le chiffrement homomorphe, par exemple, permet de réaliser des opérations de calcul sur les données chiffrées. Le déchiffrement du résultat obtenu donnera ainsi le même résultat qu'en ayant effectué ces opérations sur les données non chiffrées. Autrement dit, le résultat peut se calculer sur les données chiffrées sans avoir besoin de les déchiffrer (Fontaine & Galand, 2007).
Ce chiffrement ne supporte néanmoins que des opérations simples comme l'addition et la multiplication. Les opérations plus complexes comme la génération des applicatifs de contrôle-commande n'est pas possible par ce type de chiffrement.
D'autre part, le chiffrement fonctionnel permet à un utilisateur de décrypter et avoir accès à une fonction spécifique sur des données chiffrées. L'utilisateur a accès aux résultats de la fonction sur les données, sans pour autant avoir accès aux données (Boneh et al. 2012; O'Neill 2010; Boneh et al. 2011). En effet, si une donnée est chiffrée par une clé publique, l'algorithme décrypte la valeur d'une fonction portant sur la donnée sans avoir accès à la donnée elle-même en utilisant une clé secrète pour la fonction. D'évidence, ce type d'algorithme nécessite la connaissance de la clé publique et aussi de la clé secrète. L'utilisation de ces algorithmes sur les logiciels chiffrés n'est pas possible car on ne possède ni la clé publique utilisée pour le chiffrement des projets, ni de la clé secrète pour le déchiffrement.

2.3 Manipulation automatique des interfaces graphiques

La manipulation des interfaces graphiques consiste à produire les données chiffrées d'une manière indirecte. En effet, en réalisant des actions sur l'interface graphique du logiciel, les données chiffrées sont produites en arrière-plan.
La manipulation automatique des interfaces graphiques peut se réaliser à travers des langages de scripts, comme AutoHotKey[1] (AHK) ou AutoIt[2]. Ces langages permettent l'automatisation de tâches et l'émulation de la souris et de frappes clavier sur des interfaces graphiques sous Windows. Ainsi un script AHK peut par exemple reproduire les actions des utilisateurs (clics et frappes clavier) sur un une interface graphique.

Certaines approches proposent d'utiliser ces langages pour l'automatisation des tests (Slack, 2010). En effet, ces langages sont utilisés pour enregistrer certains scénarios de tests (Raulin et al., 2021) et aussi pour l'exécution automatique des scénarios générés sur les interfaces graphiques (Fang et al., 2021; Raulin et al., 2021). D'autre part, ces langages ont été utilisés avec succès dans l'automatisation des tâches répétitives sur des interfaces graphiques, comme les tâches d'analystes dans les laboratoires (Tentarelli et al. 2022; Rupp et al. 2022), les tâches de création des données dans des processus d'apprentissage automatique (Grigorian et al., 2020; Kretinin et al., 2021) ou dans le lancement aléatoire des scénarios d'attaques sur des systèmes industriels (Morris et al. 2015).

L'utilisation de ces langages se limite à de simples tâches d'automatisation, aucune utilisation de ces langages dans la génération des applicatifs de contrôle-commande complexes n'a été mentionnée dans la littérature.

3 APPROCHE PROPOSEE

Le but de ces travaux est de proposer une solution pour la production des données chiffrées des applicatifs de contrôle-commande.
Une première piste de solution consistait à déchiffrer les données pour ensuite les exploiter. Toutefois, les algorithmes proposés dans la littérature permettant de réaliser des calculs sur des données chiffrées sans les déchiffrer sont inapplicables sur les logiciels industriels. En effet, le chiffrement fonctionnel par exemple, nécessite d'avoir les clés publiques et secrètes,

[1] https://www.autohotkey.com/
[2] https://www.autoitscript.com/site/

utilisées dans le chiffrement. Ces clés sont définies par les constructeurs des logiciels industriels et ils nous y sont inaccessibles. Le chiffrement homomorphe reste très limité à de simples opérations d'addition et de multiplication et il est alors impossible de l'utiliser pour des opérations plus complexes comme la génération automatique ou la transformation de données. Si toutes les données d'un logiciel industriels sont chiffrées, alors la seule approche possible est d'exploiter les outils de l'interface graphique du logiciel industriel afin de générer des applicatifs dans ce logiciel industriel.

Pour y parvenir, nous avons opté pour les langages permettant la manipulation automatique des interfaces graphiques, comme AutoHotKey. Mais, l'utilisation des interfaces graphiques, qui sont spécifiques à chaque logiciel, engendre le problème de la généricité de la solution proposée. Afin de rendre notre solution générique, un flot de conception a été proposé (Figure 3). Ce flot comporte 4 étapes, détaillées dans les sections suivantes.

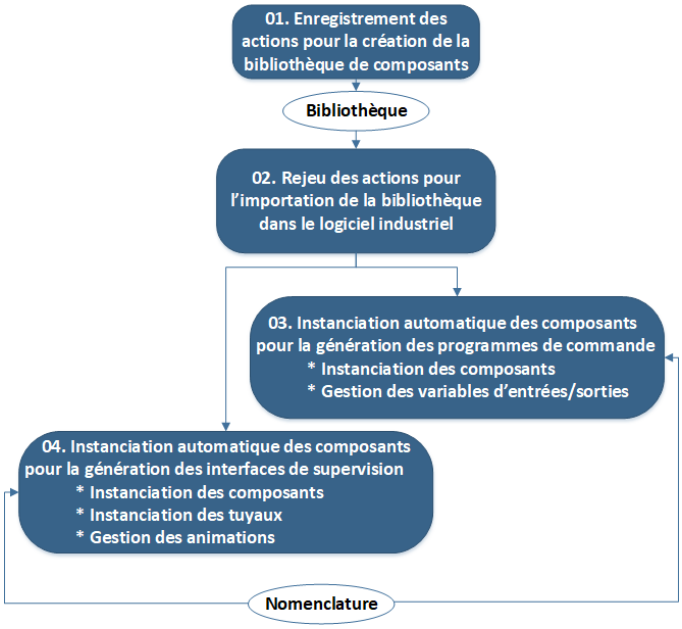

Figure 3. Solution de génération des applicatifs de contrôle-commande pour les logiciels chiffrés

3.1 Enregistrement des actions pour la création de la bibliothèque

Cette étape a pour but la création de la bibliothèque des composants en utilisant l'interface graphique du logiciel. Le défi ici est : comment capturer les connaissances nécessaires à la création des vues de commande et de supervision de chaque composant en n'utilisant que les interfaces graphiques ?

Nous nous sommes basés sur les techniques de programmation par démonstration (Goubali, 2017) pour la création de cette bibliothèque. Ces techniques reposent sur l'utilisation d'un module d'enregistrement, permettant d'enregistrer les actions de programmation, et un module de rejeu pour la reproduction des actions enregistrées.

Dans notre cas, nous avons opté pour l'enregistrement des actions manuelles (clics de souris, frappe de clavier, etc.) des concepteurs lors de la création des composants sur un logiciel donné. En effet, un module d'enregistrement des actions a été développé. Ainsi, toutes les actions manuelles au cours de la création d'un composant sont enregistrées sous la forme d'un script AHK. Pour chaque composant de la bibliothèque, un script AHK est créé sur un logiciel donné. Désormais, la bibliothèque de composants ne contient pas les vues de commande et de supervision de chaque composant (en XML) mais un script AHK qui enregistre toutes les actions pour la création de ces vues sur un logiciel donné.

3.2 Importation de la bibliothèque par le rejeu

L'importation de la bibliothèque dans le logiciel cible est réalisée par le rejeu automatique des scripts stockés dans la bibliothèque. Ce rejeu permet d'exécuter les scripts dans le but de reproduire toutes les actions manuelles nécessaires à la création du composant tant de fois que cela est nécessaire. Le module est utilisé pour l'importation de la bibliothèque au niveau de la commande et également au niveau de la supervision.

3.3 Génération des programmes de commande

Après l'importation dans le logiciel de commande de tous les composants de la bibliothèque par le rejeu de leur script, cette étape a pour but l'instanciation et le paramétrage de tous les composant de la nomenclature pour la génération des applicatifs de commande. Cette action doit être effectuée pour tous les composants listés dans la nomenclature du projet Anaxagore. C'est l'étape la plus complexe dans notre démarche.

A l'instar des logiciels de commande, le module d'instanciation sélectionne automatiquement le composant et l'instancie. Il lui attribue ensuite un nom d'instance et déclare ses variables d'entrées/sorties. Une fois le nombre d'entrées/sorties d'un composant déterminé et confirmé, le module procède à la création graphique de toutes ses variables d'entrées / sorties. Pour cela, il prend en compte le décalage dans les coordonnées X et Y pour bien les positionner sur l'interface graphique. En effet, lors de l'instanciation, le module gère les décalage X et Y afin d'éviter de positionner certains composants sur d'autres. Ce décalage est géré automatiquement par le module d'instanciation.

3.4 Génération des applicatifs de supervision

La génération des applicatifs de supervision est réalisée en trois étapes : instanciation des composants, instanciation des tuyaux et gestion des animations.

3.4.1 Instanciation des composants

L'instanciation des composants consiste à instancier les vues graphiques des composants et de les positionner sur l'interface de supervision. Pour chaque occurrence du composant dans la nomenclature, une instance de celui-ci est créée automatiquement au niveau de l'interface de supervision. La position de l'instance est calculée automatiquement à partir de ses coordonnées x et y, récupérées de la nomenclature.

Lors du positionnement des composants sur l'interfaces graphique, le problème de la mise à l'échelle a été rencontré. En effet, la génération des interfaces de supervision dépend des dimensions du matériel (écran) à utiliser. Une solution pour la mise à l'échelle automatique des composants a été proposée.

Une fois l'instance générée et positionnée, ses variables peuvent être créées dans le programme de supervision. Chaque instance possède des variables qui lui sont propres et des variables communes comme la couleur du fluide en entrée et en sortie du composant et sa propagation dans le réseau de tuyauterie. Le paramétrage de la couleur du fluide et de sa propagation se fait en associant le numéro de port avec le tuyau adjacent. La récupération des tuyaux adjacents se fait à partir de la nomenclature et des équipotentiels.

3.4.2 Instanciation des tuyaux

Les tuyaux sont de cinq types : horizontaux, verticaux, en forme de « X », des coudes en forme de « C » et enfin en forme de « T ».
L'instanciation des tuyaux consiste dans un premier temps à identifier la forme à instancier. Les tuyaux horizontaux et verticaux sont instanciés automatiquement à partir des informations contenues dans la nomenclature. Ils correspondent aux liens entre les composants. Toutefois, un calcul un peu plus complexe a été proposé afin d'identifier les tuyaux « T », « C » et « X ». Ce calcul se base sur le nombre de points en commun entre des tuyaux. En effet, s'il y a 4 tuyaux qui ont un point en commun cela veut dire qu'il faut instancier un tuyau de type « X ». Pour les tuyaux de type « C » et « T », l'angle de rotation doit aussi être calculé. Après avoir instancié les tuyaux, le module procède au paramétrage de leur couleur. Pour cela, la nomenclature est exploitée, car les tuyaux qui appartiennent au même équipotentiel ont la même couleur.

3.4.3 Gestion des animations

La gestion des animations de propagation de fluide sur les interfaces de supervision est souvent réalisée manuellement par les experts. C'est une tâche fastidieuse et itérative qui est une source d'erreur. Pour automatiser la gestion de la propagation du fluide, une stratégie de précédence a été utilisée.
Concrètement, les point de départ du circuit (comme les sources d'eau) possèdent des couleurs prédéfinies, correspondant au type du fluide. Les tuyaux sortant de ces sources recevront la même couleur de la source. Ensuite, tous les composants reliés à ces tuyaux, recevront la couleur et ainsi de suite. Pour identifier le sens de la propagation et également les composants et tuyaux à utiliser dans la propagation, les équipotentiels sont utilisés.
La Figure 4 illustre un exemple d'un équipotentiel entre les composants V2VM03, V2VM04 et H1. La couleur de la portion de tuyau (Equi19) correspond à la couleur de ses composants. Pour chaque équipotentiel identifié, une formule mathématique est générée. Ensuite, le module de gestion des animations permet d'interagir avec l'interface graphique du logiciel afin de créer automatiquement toutes les formules nécessaires à la gestion des animations.

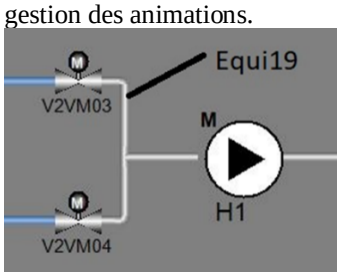

Figure 4. Gestion de la propagation de fluide

3.5 Preuve de concept & implémentation

Tous les modules de notre approche sont implémentés avec le langage AHK. Le module d'enregistrement développé pour la création des différentes vues des composants reste le même pour les différents logiciels. En effet, l'utilisateur se met sur le logiciel (par exemple Tia portal[3].) et commence la création de la vue de commande par exemple d'un composant standard. Pour donner suite à l'enregistrement de ses actions, un script AHK est généré automatiquement. Le module peut être utilisé sur différents logiciels pour la création des vues de commande ou de supervision des composants standards.
Toutefois, les modules d'importation de la bibliothèque et d'instanciation des composants sont spécifiques à chaque logiciel, car ils reposent sur la structure de son interface graphique (emplacement des boutons, des fenêtres…). Pour la preuve de concepts, nous avons développé ces modules pour trois logiciels: Straton[4], Panorama[5] et Tia portal.

4 EXPERIMENTATION

4.1 Méthode

Le but de cette expérimentation est d'évaluer l'applicabilité de notre solution dans la génération des programmes de commande et des interfaces de supervision pour des systèmes industriels. Les trois questions de recherche suivantes ont été identifiées.

- Q1. Notre approche est-elle capable de générer des programmes de commande et des interfaces de supervision pour des logiciels chiffrés ?
- Q2. Quelles sont les performances de notre approche de génération des programmes de commande par rapport à une approche basée sur l'IDM ou manuelle ?
- Q3. Quelles sont les performances de notre approche de génération des interfaces de supervision par rapport à une approche basée sur l'IDM ou manuelle ?

Pour répondre à la première question, nous avons appliqué notre approche sur le logiciel chiffré Tia.
Pour répondre aux dernières questions de recherche, nous avons comparé le temps de calcul et le nombre d'erreurs lors de la génération automatique d'un programme de commande et une interface de supervision de notre approche par rapport à l'approche IDM et à une approche manuelle. Pour cela, nous avons appliqué notre approche sur les logiciels Straton, pour la partie commande, et Panorama, pour la partie supervision. Ces deux logiciels ne sont pas chiffrés.

4.2 Système utilisé

L'analyse porte sur un système réel de traitement d'eau (Eds) à bord d'un navire (Figure 5). L'objectif de ce système est de fournir de l'eau potable aux personnels navigants (Bignon, 2012). Ce système comporte 55 composants (Tableau 1) : 14 vannes à deux voies, 5 vannes à trois voies, 3 soutes, 3 hydrophores (HP), 2 modules de traitement (TRCH et TRUV), 2 osmoseurs, des capteurs et des filtres.

4.3 Résultats

Dans ce qui suit, nous présentons les résultats obtenus pour les trois questions de validation.

4.3.1 Génération des programmes de contrôle-commande sur des logiciels chiffrés (Q1)

Comme le montre le Tableau 2, les résultats de l'application de notre approche au logiciel Tia sont encourageants. Le temps de génération de tout le projet Eds est environ de 4h. Toutefois, la génération des programmes de commande souffre de quelques erreurs. Elles viennent principalement d'une latence de réponse du logiciel qui décale les actions, réalisées sur l'interface graphique de Tia, et génère des erreurs.

[3] https://new.siemens.com/fr

[4] http ://www.copalp.com/fr/

[5] https://codra.net/fr/offre-logiciel/plateforme-supervision/logiciel-panorama-suite/panorama-e2/

Tableau 1. Liste des composants du système Eds

Composant	V2VM	V3VM	V3V	ST	HP	TRCH	TRUV	OSM	FQ	TT	FS	LS	FT	LT
Occurrence	14	3	2	3	3	1	1	2	2	4	1	3	13	3

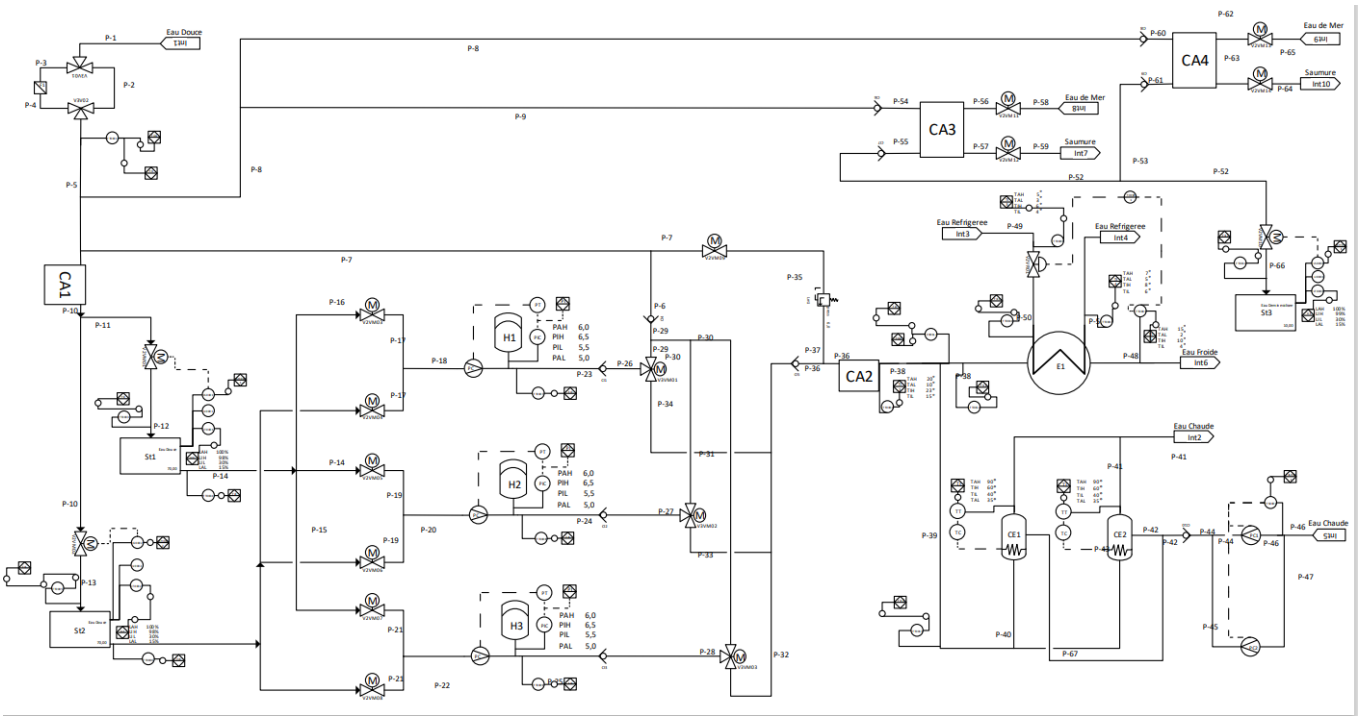

Figure 5. Le système de traitement d'eau Eds (Bignon, 2012)

En effet, lors de la création d'un composant volumineux (comme le composant LT, par exemple), le logiciel industriel prend plus de temps que sur les petits composants et son temps de réponse augmente. Pour la réduction de ces latences, nous avons augmenté le temps d'attente entre les différentes actions de génération.

Il est à noter que la génération par IDM est impossible pour le logiciel Tia car il est chiffré.

Tableau 2. Résultats de la génération des programmes de commande sur Tia

Tia	Temps importation	Temps instanciation	Erreurs	Type erreurs
IDM	x	x	x	x
Notre approche	2h12	2h	1 (Instanciation) 2 (Importation)	Latence

4.3.2 Performance de génération de commande de notre approche vs une approche IDM ou manuelle (Q2)

Le Tableau 3 présente les performances, en termes de temps de calcul et de nombre d'erreurs générées, des trois approches : manuelle, IDM et la nôtre lors de la génération des programmes de commande du système Eds sur Straton. L'approche IDM reste la plus performante en terme du nombre d'erreurs et en temps de calcul. En effet, la génération et l'instanciation de tous les composants du système Eds ont été réalisées en quelques secondes. Notre approche a pris quelques heures car elle utilise les interfaces graphiques des logiciels pour la génération. Néanmoins, elle reste plus rapide qu'une approche manuelle. En effet, un gain de 23% en termes de temps de calcul a été constaté.

Tableau 3. Résultats de la génération des programmes de commande sur Straton

Approche	Temps import	Temps instanciation	Erreurs	Type erreurs
IDM	0,173s	0,338s	0	0
Notre approche	2h	1h45	0	0
Manuelle	2h	2h18	2	Nom de variables

4.3.3 Performances de génération d'interface de notre approche vs une approche IDM ou manuelle (Q3)

Pour la génération de l'interface de supervision du système Eds sur le logiciel Panorama, l'approche par IDM avec un temps de génération de 2 secondes, reste la plus performante, comparée à notre approche et à une approche manuelle (Tableau 4). Toutefois, notre approche est plus performante par rapport à une approche manuelle surtout sur l'étape d'instanciation des composants (un gain de 95% en temps de calcul). En effet, l'étape de l'importation de la bibliothèque est un rejeu des actions du concepteur, ce qui explique que le temps d'importation de la bibliothèque dans notre approche est le même que celui dans une approche manuelle.

Cependant, sur la partie instanciation des composants et des tuyaux, notre approche est très performante par rapport à une approche manuelle. Nous avons constaté aussi que dans une approche manuelle, les concepteurs commettent certaines erreurs surtout au niveau de la gestion des animations, alors

que notre approche permet de générer toutes les équations des animations sans aucune erreur.

Tableau 4. Résultats de la génération des programmes d'interface sur Panorama E2

Approche	Temps import	Temps instanciation	Erreurs	Type erreurs
IDM	2s	18s	0	0
Notre approche	7h	1h17	0	0
Manuelle	7h	21h	10	Equations d'animation

4.4 Discussion

En résumé, dans le cadre de la génération des programmes de commande et des interfaces de supervision pour des systèmes de contrôle-commande, si les logiciels cibles ne sont pas chiffrés, l'approche par IDM reste la plus adaptée et la plus performante. Toutefois, celle-ci n'est pas applicable sur des logiciels chiffrés. Sur des logiciels chiffrés, notre approche est la seule solution et elle est plus performante qu'une approche manuelle avec un gain en temps de calcul de 23% pour la génération de commande et de 95% pour la génération d'interface de supervision, comparée à une approche manuelle.

Malgré les résultats encourageants de notre approche, celle-ci souffre de certaines limites. En effet, les successions d'actions courtes effectuées sur un logiciel provoquent des latences non détectées. Nos travaux futurs porteront sur l'amélioration de ces performances. De plus, les propriétés graphiques sont parfois différentes entre les logiciels cibles ce qui se traduit par des parties des scripts utilisés spécifiques à chaque logiciel. Il est alors important de traiter l'aspect générique de notre solution afin d'étendre sa transférabilité.

5 CONCLUSION

Une nouvelle solution a été proposée pour la génération des applicatifs de contrôle-commande aux logiciels chiffrés. Cette solution utilise les interfaces graphiques des logiciels pour la génération. La manipulation automatique des interfaces graphiques produit les données chiffrées en arrière-plan.
Toutes les actions sur les interfaces graphiques, nécessaires à la production de ces données ont été automatisées. Toutefois, l'utilisation des interfaces graphiques, qui sont spécifiques à chaque logiciel, engendre le problème de la généricité de la solution proposée. Nous avons alors proposé un flot de conception qui se composent de deux modules : module de création d'une bibliothèque de composants et un module d'instanciation de ces composants. Le premier module enregistre et rejoue toutes les actions (clics, frappes de clavier…) sur l'interface graphique du logiciel pour la création des différents composants. Ensuite, le deuxième module reproduit toutes les actions nécessaires à l'instanciation d'un composant dans le logiciel, tout en mettant à jour les informations du composant (nom d'instance, nom des variables d'entrées/sorties…). Au niveau de la supervision, des solutions ont été proposées pour la gestion automatique des animations et aussi la mise à l'échelle des composants sur les interfaces graphiques.
Une expérimentation a été conduite afin de comparer les performances, en termes de précision de la génération et de temps de calcul, entre cette solution et une approche basée sur l'IDM. Sur des logiciels chiffrés notre solution a été appliquée avec succès contrairement à une approche basée sur l'IDM qui reste inapplicable sur ces logiciels. Sur des logiciels non-chiffrés, les résultats montrent que l'approche basée sur l'IDM est plus performante car la génération est réalisée en quelques secondes alors que notre nouvelle approche prend quelques heures. Néanmoins, comparée à une approche manuelle, notre solution présente un gain en temps de calcul de 23% pour la génération de commande et de 95% pour la génération de l'interface de supervision. Elle permet également d'automatiser des processus de gestion des animations et ainsi éviter des erreurs, commises dans une approche manuelle.

D'autre part, notre solution ne permet actuellement que la génération des applicatifs de contrôle-commande élémentaires. Une extension de celle-ci est nécessaire pour la génération des applicatifs de contrôle-commande de haut niveaux (fonctions).

6 REMERCIMENT

Les auteurs de cet article tiennent à remercier Johann FARRUGIA pour tous les codes qu'il a développé.

7 REFERENCES

Bato, T., Thomas, G., & Varela, F. (2017). A Model-driven Generator to Automate the Creation of HMIs for the CERN Gas Control Systems.

Bignon, A. (2012). Génération conjointe de commandes et d ' interfaces de supervision pour systèmes sociotechniques reconfigurables [PhD Thesis]. Université de Bretagne Sud.

Bignon, A., Berruet, P., & Rossi, A. (2010). Joint generation of controls and interfaces for sociotechnical and reconfigurable systems. 2010 IEEE International Conference on Systems, Man and Cybernetics, 749–755.

Bignon, A., Rossi, A., & Berruet, P. (2013). An integrated design flow for the joint generation of control and interfaces from a business model. Computers in Industry, 64(6), 634–649.

Boneh, D., Sahai, A., & Waters, B. (2011). Functional encryption: Definitions and challenges. Theory of Cryptography Conference, 253–273.

Boneh, D., Sahai, A., & Waters, B. (2012). Functional encryption: A new vision for public-key cryptography. Communications of the ACM, 55(11), 56–64.

Combemale, B. (2008). Meta modeling Approach for Model Simulation and Verification: Application To Process Engineering. French Phd Thesis,(IRIT, Enseeiht).

De Lamotte, F. F. (2006). Proposition d'une approche haut niveau pour la conception, l'analyse et l'implantation des systèmes reconfigurables. LESTER, Université de Bretagne Sud, Lorient.

Drath, R., Fay, A., & Schmidberger, T. (2006). Computer-aided design and implementation of interlock control code. 2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2653–2658. https://doi.org/10.1109/CACSD-CCA-ISIC.2006.4777057

Fang, D., Song, Z., Guan, L., Liu, P., Peng, A., Cheng, K., Zheng, Y., Liu, P., Zhu, H., & Sun, L. (2021). ICS3Fuzzer: A Framework for Discovering Protocol Implementation Bugs in ICS Supervisory Software by Fuzzing. Annual Computer Security Applications Conference, 849–860.

Fontaine, C., & Galand, F. (2007). A survey of homomorphic encryption for nonspecialists. EURASIP Journal on

Information Security, 2007, 1–10.
Goubali, O. (2017). Apport des techniques de programmation par démonstration dans une démarche de génération automatique d'applicatifs de contrôle-commande [PhD Thesis]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers.
Goubali, O., Bignon, A., Berruet, P., Girard, P., & Guittet, L. (2014). Anaxagore, un exemple d'ingénierie dirigée par les modèles pour la supervision industrielle. Ergonomie et Informatique Avancée (ERGO'IA 2014).
Goubali, O., Girard, P., Guittet, L., Bignon, A., Kesraoui, D., Berruet, P., & Bouillon, J.-F. (2016). Designing functional specifications for complex systems. International Conference on Human-Computer Interaction, 166–177.
Goubali, O., Girard, P., Guittet, L., Bignon, A., Kesraoui, D., Boulhic, L., Berruet, P., & Bouillon, J.-F. (2016). Evaluation of tool support for functional specification of complex systems. Proceedings of the 15th Ergo'IA" Ergonomie Et Informatique Avancée" Conference, 1–8.
Grigorian, A., Fang, P., Kirk, T., Efendizade, A., Jadidi, J., Sighary, M., & Cohen-Addad, D. I. (2020). Learning from gamers: Integrating alternative input devices and AutoHotkey scripts to simplify repetitive tasks and improve workflow. Radiographics, 40(1), 141–150.
Gruner, S., Weber, P., & Epple, U. (2014). Rule-based engineering using declarative graph database queries. 2014 12th IEEE International Conference on Industrial Informatics (INDIN), 274–279. https://doi.org/10.1109/INDIN.2014.6945520
Jbair, M., Ahmad, B., Mus'ab H, A., Vera, D., Harrison, R., & Ridler, T. (2019). Automatic PLC code generation based on virtual engineering model. 2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS), 675–680.
Koziolek, H., Burger, A., Platenius-Mohr, M., & Jetley, R. (2020). A classification framework for automated control code generation in industrial automation. Journal of Systems and Software, 166, 110575.
Kretinin, O. V., Popov, E. V., Tsapaev, A. P., Fedosova, L. O., & Tyurikov, M. I. (2021). Synthesis and Visualization of Image Datasets of Parametric 3D Model for Neural Network Training and Testing in Data-Poor Conditions. Scientific Visualization, 13(5). https://doi.org/10.26583/sv.13.5.06
Morris, T. H., Thornton, Z., & Turnipseed, I. (2015). Industrial control system simulation and data logging for intrusion detection system research. 7th Annual Southeastern Cyber Security Summit, 3–4.
O'Neill, A. (2010). Definitional issues in functional encryption. Cryptology EPrint Archive.
Qasim, I., Anwar, M. W., Azam, F., Tufail, H., Butt, W. H., & Zafar, M. N. (2020). A model-driven mobile HMI framework (MMHF) for industrial control systems. IEEE Access, 8, 10827–10846.
Raulin, V., Gimenez, P.-F., Han, Y., Tong, V. V. T., & Ouairy, L. (2021). GUI-Mimic, a cross-platform recorder and fuzzer of Graphical User Interface. GreHack 2021.
Rupp, N., Peschke, K., Köppl, M., Drissner, D., & Zuchner, T. (2022). Establishment of low-cost laboratory automation processes using AutoIt and 4-axis robots. SLAS Technology, 27(5), 312–318.
Sacha, K. (2010). Verification and implementation of software for dependable controllers. International Journal of Critical Computer-Based Systems, 1(1–3), 238–254.
Slack, J. M. (2010). System testing on the cheap. 2010 Information Systems Educators Conference Proceedings.
Steinegger, M., & Zoitl, A. (2012). Automated code generation for programmable logic controllers based on knowledge acquisition from engineering artifacts: Concept and case study. Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012). https://doi.org/10.1109/ETFA.2012.6489546
Tentarelli, S., Romero, R., & Lamb, M. L. (2022). Script-based automation of analytical instrument software tasks. SLAS Technology, 27(3), 209–213.
Wang, R., Gu, M., Song, X., & Wan, H. (2009). Formal specification and code generation of programable logic controllers. 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, 102–109.